SUPRA: a sampling-query optimization method for large-scale OLAP

Ushijima, K. Fujiwara, S. Nishizawa, I. Sagawa, N.
Central Res. Lab., Hitachi Ltd., Tokyo;

This paper appears in: **Database and Expert Systems Applications, 1998. Proceedings. Ninth International Workshop on**
08/26/1998 -08/28/1998, 25-28 Aug 1998
Location: Vienna , Austria
On page(s): 232-237
25-28 Aug 1998
References Cited: 12
IEEE Catalog Number: 98EX130
Number of Pages: xix+1023
INSPEC Accession Number: 6034651

Abstract:
Relational online analytical processing (ROLAP) reduces the amount of storage required for maintaining various sizes of data cubes by materializing only parts of them in a lazy evaluation manner. In ROLAP however, cube creation queries need to be issued repeatedly in order to search for useful features (i.e. rules or patterns) within large scale databases. The cube creation cost can be a bottleneck in the whole ROLAP processing. The cost of the queries can be effectively reduced by estimating the query results using samples. To maintain the accuracy of ROLAP even when using samples, the samples need to be extracted in an appropriate unit. However, conventional query optimization methods only support record based sampling and cannot be applied for complex queries that have other sampling units, such as the ones that include grouping aggregate operations. We develop a query optimization method named SUPRA that preserves the sampling unit used in random data extraction. The method is designed to preserve both the sampling unit and the randomness of the sampling operation. Using this method, typical ROLAP queries can be transformed into more efficient ones than those obtained through conventional methods

Index Terms:
query processing relational databases transaction processing very large databases ROLAP processing ROLAP queries SUPRA aggregate operations complex queries conventional query optimization methods cube creation queries data cubes large scale OLAP large scale databases lazy evaluation query results random data extraction record based sampling relational online analytical processing sampling operation sampling query optimization method sampling units

SUPRA: A Sampling-query Optimization Method for Large-scale OLAP

Kazutomo Ushijima, Shinji Fujiwara, Itaru Nishizawa, and Nobutoshi Sagawa
Central Research Laboratory, Hitachi, Ltd. Tokyo 185-8601, Japan
{ushijima,fujiwara,itaru,sagawa}@crl.hitachi.co.jp

Abstract

Relational online analytical processing (ROLAP) reduces the amount of storage required for maintaining various sizes of data cubes by materializing only parts of them in a lazy evaluation manner. In ROLAP, however, cube-creation queries need to be issued repeatedly in order to search for useful features (i.e. rules or patterns) within large-scale databases. The cube-creation cost can be a bottleneck in the whole ROLAP processing. The cost of the queries can be effectively reduced by estimating the query results using samples. To maintain the accuracy of ROLAP even when using samples, the samples need to be extracted in an appropriate unit. However, conventional query-optimization methods only support record-based sampling and cannot be applied for complex queries, that have other sampling units, such as the ones that include grouping aggregate operations. In this paper, we develop a query-optimization method named SUPRA that preserves the sampling unit used in random data extraction. The method is designed to preserve both the sampling unit and the randomness of the sampling operation. Using this method, typical ROLAP queries can be transformed into more efficient ones than those obtained through the conventional methods.

1. Introduction

In recent years, the importance of data warehouse (DW) systems has been widely recognized, and many large-scale DW systems have been built for business purposes. Online analytical processing (OLAP) [1, 4, 5] is one of the most practical analysis methods used with the DW systems. The OLAP tries to find useful features from the multi-dimensional summary data usually called data cubes. Since a cube is prepared for each combination of analytical attributes, a large amount of storage is required for keeping various sizes of data cubes in advance. Relational-OLAP (ROLAP) is an effective approach to reduce the amount of storage by materializing parts of the data cubes in a lazy evaluation manner. In practical situations, however, in order to obtain data-cubes including useful features, appropriate combinations of analytical attributes and also appropriate preconditions for the records of the data cubes should be determined. Therefore, the ROLAP approach requires a number of cube-creation queries to be issued repeatedly in order to find appropriate features within the databases [3]. To make the entire analysis process efficient, each cube-creation query should be processed rapidly.

One promising approach to enable rapid cube creation is to use sampling instead of using the whole database. The analyst can estimate the result of the query using sampled data and determine the parameters of the next cube-creation query based on the estimation. The reduction of processing data shortens the processing time of each query and the analyst can obtain the target features more quickly. In optimization of sampling queries, the processing time can be reduced drastically by applying the sampling operation at an early stage of the query processing. However in business DW systems, because the data is shared among the various offices, the unit of the record used for storing does not always agree with the unit of the record needed for the analysis. For instance, suppose we have a database that includes a sales table as shown in Figure 1.

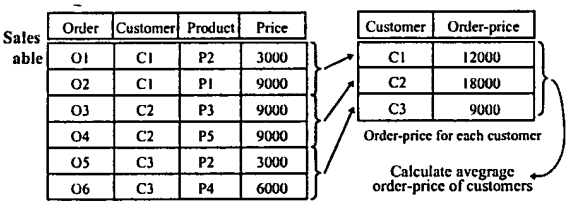

Figure 1. Conversion of the unit of records

In this table, the unit of stored records is the same as the unit of an order. If we want to estimate the average order-price of all customers, we cannot simply sample the records from this table. The records having the same customer number need to be sampled together as one unit, and the average should be estimated using these records. However, the conventional query optimization methods [7, 8, 10] only support record-based sampling and do not consider the *sampling unit* of the sampling operation. Because a grouping aggregate operation converts the unit of records in its processing, the conventional methods cannot be applied for the queries including the grouping aggregate operations. This

limitation of the conventional methods prevents further optimization of the sampling queries in practical use.

In this paper, we propose a new query optimization method named SUPRA (sampling unit preservation method) which preserves the sampling unit during the optimization of the sampling operations. This method enables preservation of the sampling unit by adopting a special sampling operation in which all the records specified as included in the same sampling unit are guaranteed to be extracted all together within the resulting samples. The method preserves both the sampling unit and the randomness of data extraction through its query optimization. When the cube-creating sampling queries are issued to the database, the query optimization module of the database system automatically identifies the sampling units of the queries and transforms them into equivalent ones in which the application point of the sampling operations have been moved forward.

2. Sampling units of sampling queries

In business DW systems, various kinds of operational data are collected from throughout the company, and the collected data are shared among offices having different purposes on data analysis. A sampling operation used for the analysis must extract records in an unit suitable for the purpose of the analysis. For instance, a query (Figure 2) is issued to the table (Figure 1) in order to estimate the average order-price of the customers. The RANDOM keyword in the query indicates the AVG operation should be processed using samples. In this example, the records having the same customer number need to be sampled all together as one sampling unit of the sampling operation (Figure 3(a)).

```
SELECT AVG( RANDOM(OrderPrice) )
FROM SELECT Customer, SUM(Price) AS OrderPrice
        FROM SalesTable
        GROUP BY Customer;
```

Figure 2. Sampling query

Unfortunately, the conventional methods [7, 8, 10] assume record-based sampling for its optimization and cannot be applied to queries that include operations other than record-based sampling. If the sampling unit does not agree with the unit for analysis, a proper estimation is no longer expected from the sampled records. Suppose we sample two customers at random from the above table and sum up the prices of them then calculate the average of both customers. The expected value of the average order-price is 13000. This is the correct answer. However, if we apply record-based sampling to the table (Figure 3(b)) and sample four orders (two orders for each two customers) from the table at random, the expected value of this case is 9533.3$\cdots$. As shown like this, because the semantics of the sampling operation no longer agrees with the purpose of the analysis, we cannot draw the correct estimation from this approach. A method for sampling-query optimization that can preserve the sampling unit is needed to realize accurate sampling-based ROLAP processing.

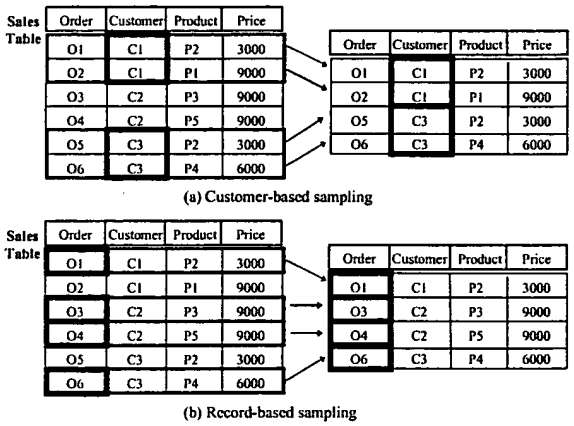

Figure 3. Appropriate sampling unit

3. SUPRA: preservation of the sampling units

In our approach, in order to preserve a sampling unit through the query optimization, a special sampling operation is introduced as a new basic operation. In the following, we first define the semantics of this sampling operation, then we explain query transformation procedures of the sampling operation with other basic database operations. We also give an example of the query optimization and an efficient random sampling technique from a database table. We represent the table that results from an operation F to a table T by an execution formula F(T).

3.1. Definition of the sampling operation

SAMPLING operation: In our SAMPLING operation S(SC,SGC,T), two parameters are introduced to keep a sampling unit during the query transformation : sampling column (SC) and sampling grouping column (SGC). The semantics of the operation is defined by these parameters.

(i) If the sampling column is specified as a list of column names, the operation assigns a value randomly for each sampling columns and extracts all the records having the specified values in their sampling columns.

(ii) If the sampling column is specified as "not specified", the operation extracts the records on a record basis.

(iii) If the sampling column becomes an empty set, the operation extracts all the records from the table.

The sampling operation is repeatedly applied to the table until sufficient number of samples are obtained.

In addition, another set of column names can be specified as the sampling grouping column (SGC) of the SAMPLING operation. In that case, the records in table T are classified

into groups according to the value of the sampling grouping column, and records within each group having the same value in their sampling columns are regarded as one sampling unit. The extraction probability of each sampling unit needs to be the same within each group, while the probability can differ among the groups (see Figure 4).

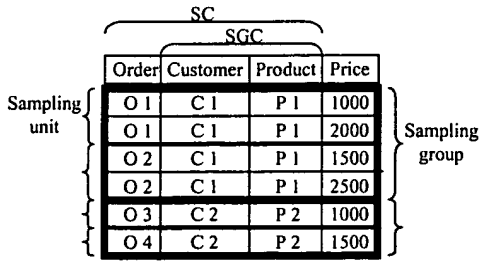

Figure 4. Sampling parameters

3.2. Query optimization procedures

In general, a sampling query can be optimized by pushing down the sampling operation and reducing the number of records to be processed. In the following, formulas for query transformation procedures of basic database operations are described. In each description, ≡ denotes equality of sampling operations on both sides of the formula. We define the equivalence of the sampling operations as satisfying following two conditions through the query transformation.

(C1) The sampling unit is preserved.

(C2) The extraction probability of each sampling unit is preserved.

The query transformation begins with insertion of a sampling operation into the sampling query just before its aggregate operation. At the insertion time, the SC of the operation is specified as "not specified", and the SGC is specified as the same as the grouping column of the aggregate operation (if it exists). In the following, proofs are given for the cases when the SC is specified other than "not specified". The proofs for the record-based sampling are already given in the previous work [7, 8, 10] (except for the grouping aggregate operation).

Definition: $R_{sc,sgc}^{L(or R)}$ denotes a set of records extracted as one sampling unit by the execution of the left (or right) side of the formula assuming the sampling parameters are given as SC = *sc* and SGC = *sgc*. (*sc* and *sgc* are lists of randomly specified values.)

(1) PROJECT operation

The PROJECT operation P(PC, T) extracts the projection columns (PC) from table T. In this paper, we assume that the operation does not eliminate duplicates. A query including the PROJECT operation can be optimized by simply exchanging its application order.

$$S(SC, SGC, P(PC, T)) \equiv P(PC, S(SC, SGC, T))$$

Proof: (C1) Suppose a record $r \in R_{sc,sgc}^{L}$. Because r satisfies SC = *sc*, r is also extracted in the right side of the formula (here we assume SC ⊆ PC) (i.e. $R_{sc,sgc}^{L} \subseteq R_{sc,sgc}^{R}$). The same holds true for the right side (i.e. $R_{sc,sgc}^{L} \supseteq R_{sc,sgc}^{R}$). Thus it follows that $R_{sc,sgc}^{L} = R_{sc,sgc}^{R}$.
(C2) From the discussion above, a sampling unit and the value of the SC have one-to-one correspondence. Thus if we apply the same sampling procedure for both sides of the formula, the extraction probability of the sampling units will be preserved.

(2) SELECT operation

The SELECT operation C(Pred, T) extracts records satisfying a selection condition *Pred* from table T. A query including the SELECT operation can be optimized by simply exchanging its application order.

$$S(SC, SGC, C(Pred, T)) \equiv C(Pred, S(SC, SGC, T))$$

Proof: (C1) Suppose a record $r \in R_{sc,sgc}^{L}$. Because r satisfies SC = *sc* and the condition *Pred*, r is also extracted in the right side of the formula. The same holds true for the right side. Thus it follows that $R_{sc,sgc}^{L} = R_{sc,sgc}^{R}$.
(C2) The same reason with the previous operation.

(3) JOIN operation

The JOIN operation J(JC, S, T) joins two records from tables S and T, if they have the same value for the join column (JC). Here we assume equi-join. A query including the JOIN operation can be optimized by exchanging their application order and limiting the sampling columns and the sampling grouping columns to respective columns also included in each of the tables.

$$S(SC, SGC, J(JC, S, T)) \equiv J(JC, S(SC/S, SGC/S, S), S(SC/T, SGC/T, T))$$

Proof: (C1) Suppose a record $r \in R_{sc,sgc}^{L}$. The record r can be divided into two records having the same JC value by limiting the columns of r to respective columns included in each of the tables. Because these records satisfies the restriction on the SC values of the right side, the records also extracted and joined in the right side of the formula. The similar discussion also holds true for the right side. As a result, we can conclude that $R_{sc,sgc}^{L} = R_{sc,sgc}^{R}$.
(C2) The same reason with the previous operation.

In this query transformation, in order to improve the join selection ratio, the same SC values should be specified in both of the sampling operations distributed to the respective tables.

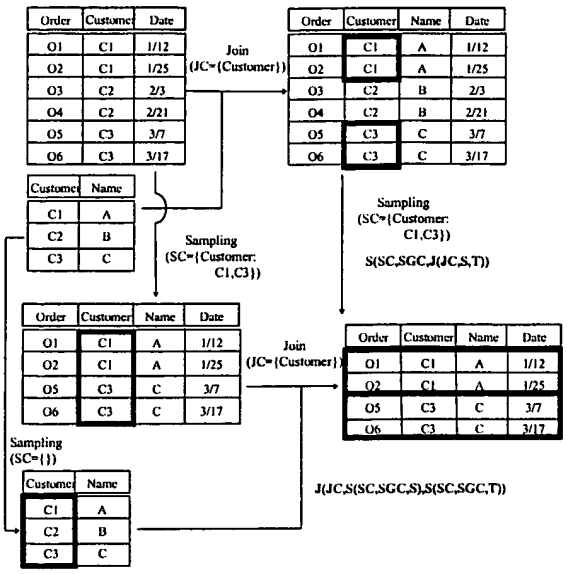

Figure 5. Transformation of JOIN query

(4) GROUPING AGGREGATE operation

The GROUPING AGGREGATE operation A(AC, GC, T) classifies the records of the table T into groups according to the value of grouping columns (GC), then calculates the aggregate value (such as summation or average) of the aggregate columns (AC) for each group. A query including the GROUPING AGGREGATE operation can be optimized by simply exchanging its application order if SC ⊆ GC (except the case when the SC is specified as "not specified").

$$S(SC, SGC, A(AC, GC, T)) \equiv A(AC, GC, S(SC, SGC, T))$$

Proof: **(C1)** The grouping aggregate operation does not change the SC value if SC ⊆ GC. Suppose a record $r \in R^L_{sc,sgc}$. Because r satisfies SC = sc, the record also extracted in the right side of the formula. The similar discussion also holds true for the right side. Thus it follows that $R^L_{sc,sgc} = R^R_{sc,sgc}$.
(C2) The same reason with the previous operation.

When the value of the sampling columns specified as "not specified", the resulting records of grouping aggregation are sampled on a record basis. Because the unit of a record changes during this operation, the conventional methods cannot cope with this case. The records having the same value in the grouping columns should be extracted as one sampling unit. In this case, we will apply the previous transformation procedure regarding the grouping columns as the sampling columns (Figure 6). Because each record in the results of the grouping aggregate operation and the value of the grouping column have one-to-one correspondence, the sampling unit of the sampling query can be handled correctly by this conversion.

In addition, if the sampling grouping columns (SGC)

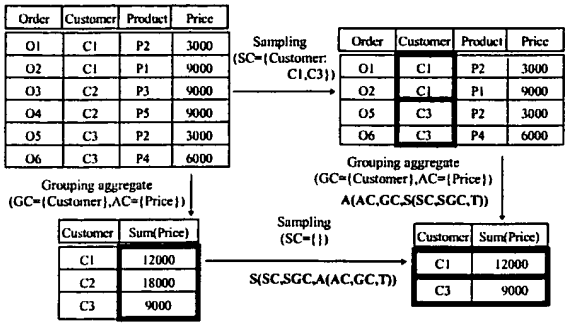

Figure 6. Transformation of GROUPING AGGREGATE query

are specified and satisfies SGC ⊆ SC, we can substitute the sampling columns of the operation with the difference between the sampling columns and the sampling grouping columns (SC-SGC) (Figure 7).

$$S(SC, SGC, T) \equiv S(SC - SGC, SGC, T)$$

Decrease in the number of sampling columns reduces the CPU cost at the data extraction time. In the following, we will examine the equivalence of the sampling unit within a group constructed by the sampling grouping columns.
Proof: **(C1)** Suppose a record $r \in R^L_{sc,sgc}$. Because r satisfies SC = sc and SGC = sgc, the record also satisfies the restriction on the SC values of the right side. Thus the record is also extracted and classified into the same group in the right side of the formula. Similarly, if we assume $r \in R^R_{sc,sgc}$, then r satisfies the restriction on the SC values except for the SGC values (SC-SGC=sc-sgc), and also satisfies SGC = sgc. From the above two facts, it follows that r satisfies SC = sc. Therefore, the record r also extracted and classified into the same group in the left side of the formula. Hence we can conclude that $R^L_{sc,sgc} = R^R_{sc,sgc}$.
(C2) The same reason with the previous operation.

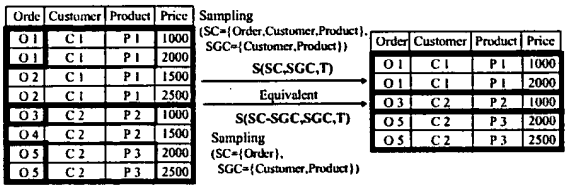

Figure 7. Transformation of SAMPLING query

3.3. An example of query transformation

This subsection shows an example of the query transformation. Suppose a database consists of three tables : a customer table, an order table, and a product table. These tables store the information generated through a typical sales management work. The custID column is the key column of

the customer table. The orderID column is the key column of the order table, and the custID column of the order table is a foreign key column of the customer table. The orderID column of the product table is a foreign key column of the order table. Then, suppose a query shown in Figure 8 is issued to the database to estimate the average order-price of customers for each customer category. The execution formula for this query is given as Figure 9.

```
SELECT Segment, Priority, Shipmode, AVG(RANDOM(OrderPrice))
FROM SELECT OrderID, Segment,
                Priority, Shipmode,SUM(Price)AS OrderPrice
        FROM CustomerTable, OrderTable, ProductTable
        WHERE CustomerTable.CustomerID = OrderTable.CustomerID
            AND OrderTable.OrderID = ProductTable.OrderID
        GROUP BY OrderID, Segment, Priority, Shipmode
GROUP BY Segment, Priority, Shipmode;
```

Figure 8. Complex sampling query

```
A(OrderPrice, {Segment, Priority, Shipmode},
  S(NotSpecified, {Segment, Priority, Shipmode},
    A(Price, {OrderID, Segment, Priority, Shipmode},
      J({OrderID}, ProductTable,
        J({CustomerID}, CustomerTable, OrderTable))))
```

Figure 9. Execution formula for the query

When we apply our optimization method to the example, the application order of the sampling operation is exchanged with the grouping aggregate operation and the join operations step-by-step. Figure 10 shows the execution formula of the transformed example query. The sampling operation is moved beyond the operations and is applied directly to the product and order tables. This reduces the number of records to be processed drastically.

```
A(OrderPrice, {Segment, Priority, Shipmode},
  A(Price, {OrderID, Segment, Priority, Shipmode},
    J({OrderID}, S({OrderID}, {Segment, Priority, Shipmode}, ProductTable),
      J({CustomerID}, CustomerTable,
        S({OrderID}, {Segment, Priority, Shipmode}, OrderTable)))))
```

Figure 10. Transformed query

3.4. Efficient random sampling

After the query optimization, sampling operations are applied directly to the tables. To extract records from the tables efficiently while preserving the sampling units, we adopted a cluster sampling method using hash functions. In this method, several "partition columns" are specified for each table in advance and all records in the table are classified into buckets according to their hash values of the partition columns. To each of the buckets, several blocks (fixed-size continuous areas) of the disk are allocated, and the records in a bucket are stored in these blocks. At the time of data extraction, a hash value is selected randomly for each of the sampling columns and all the records in a bucket having the specified hash value are extracted from the database as samples. If the partition columns and the sampling columns are the same, we can extract records from the buckets instead of evaluating hash values of the records. When some sampling columns do not agree with the partition columns, the hash functions are used on those remaining columns. If the sampling column is specified as "not specified", the records are sampled an a record basis using conventional methods. For instance, the OrderID column can be used as a partition column for the example given in the previous subsection. Using this method, we do not have to evaluate hash values of all the records nor have to access the table randomly using an index.

Data size	1 GB (10M records)
Number of nodes	4
Size of main memory	256 MB/Node
Number of partitions	64 Buckets/Node

Table 1. Experimental Parameters

4. Performance issues

This section presents the experimental results of applying our SUPRA method to a typical ROLAP query. We have implemented our SUPRA method on the HiRDB (a commercial parallel database system of Hitachi) running on the SR4300 (SP2 comparable) share-nothing-type parallel computer (CPU: Power2, 66 MHz) executing the AIX 4.1 OS. We used TPC-D [12] as a data model for the experiments. The query shown in Figure 8 was used as an example query. Experimental parameters are listed in Table 1. We compared the following four sampling methods.

1) **Conventional method**
   - (a) **Without an index** Samples are extracted from the order table and the product table independently, and the records are joined with the customer table using a sort-merge method. However, this method estimate the query result incorrectly.
   - (b) **With indexes** Samples extracted from the order table are joined with the product table and the customer table via indexes using a nest-loop method. In this method, because the sampling unit of the query is guaranteed by the indexes, the correct estimation can be obtained.

2) **SUPRA method**
   - (a) **Optimization only** Samples are extracted from the order table and the product table according to the hash value of the records, and the resulting records are joined with the customer table using a sort-merge join.
   - (b) **Optimization + hash partitioning** Samples are extracted from the order table and the product table according to the hash values of the buckets,

and the resulting records are joined with the customer table using a sort-merge join.

The following graph shows the relative sample extraction time comparing to the time when no sampling operation was included in the query. Figure 11 indicates that our query optimization method (method 2(a)) reduces the query processing time to about half of that when using the conventional method without an index (method 1(a)). When we combine our hash partitioning method to the previous optimization method, the resulting SUPRA method (method 2(b)) is twice as fast as the conventional method using the indexes (method 1(b)). Over 10,000 records are typically required for a multi-dimensional OLAP analysis (assuming the required accuracy is 99%). Our results confirm the superiority of our SUPRA methods for typical ROLAP queries.

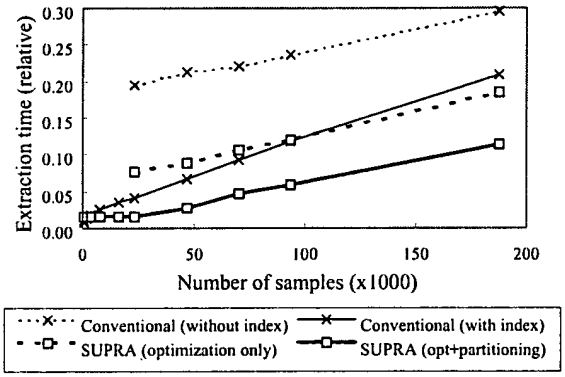

Figure 11. Execution time of record sampling

5. Related work

Hellerstein et al. [6] tried to process online aggregation queries by using sampling, and they claimed that accurate computation of the entire data cube is not necessary; approximations of the aggregates seem to be sufficient in many situations. They do not mention about the query optimization of sampling queries. On the other hand, a number of sampling methods [2, 11, 9] to extract records from database files have been proposed, however, these methods do not consider the relational queries.

The most immediately related method of our approach is described in the literature [7, 8, 10]. This conventional method provides query transformation procedures that exchange the application order between the sampling operation and each of the basic database operations, which only preserve the record-level extraction probability. In this method, the sampling units of the sampling operations are not considered and the method cannot be applied to sampling queries including record- unit converting operations, such as grouping aggregate operations. On the contrary, our query optimization method is designed to preserve the sampling unit during the query optimization, and can be used when the unit of storage and the unit of analysis do not agree. In addition, our method adopts a hash partitioning to extract sample records efficiently from a database table.

6. Conclusion and future work

We have developed a query optimization method, which preserves the sampling unit of the sampling operation during the query transformation, and a hash partition method to extract sample records efficiently form a database table. We then presented our experimental results from the evaluation of our developed methods. Using these methods, practical cube-creating queries can be processed rapidly.

Future work of our method is to provide a quantitative method for determining the optimal partition columns. Using our method, we are planning to improve the interactivity of our OLAP products.

References

[1] S. Agarwal, R. Agrawal, et al. On the computation of multidimensional aggregates. In *Proc. of the 22nd VLDB*, pages 506–521, Bombay, India, September 1996.

[2] G. Antoshenkov. Random sampling from pseudo-ranked b+ trees. In *Proc. of the 18th VLDB*, pages 375–382, Vancouver, Canada, August 1992.

[3] U. M. Fayyad et al. *Advances in Knowledge Discovery and Data Mining*. AAAI Press, 1996.

[4] J. Gray, A. Bosworth, et al. Data cube: A relational aggregation operator generalizing group-by, cross-by, and sub-total. In *Proc. of the 12th ICDE*, pages 152–159, New Orleans, USA, February 1996.

[5] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 205–216, Montreal, Canada, June 1996.

[6] J. M. Hellerstein, P. Haas, and H. Wang. Online aggregation. In *Proc. of ACM SIGMOD*, pages 171–182, Arizona, USA, May 1997.

[7] F. Olken. *Random Sampling from Databases*. PhD thesis, University of California, Berkeley, 1993.

[8] F. Olken and D. Rotem. Simple random sampling from relational databases. In *Proc. of the 12th VLDB*, pages 160–169, Kyoto, Japan, August 1986.

[9] F. Olken and D. Rotem. Random sampling from b+ trees. In *Proc. of the 15th VLDB*, pages 269–277, Amsterdam, The Netherlands, August 1989.

[10] F. Olken and D. Rotem. Random sampling from database files: A survey. In *Proc. of the 5th International Conference on Statistical and Scientific Databases Management*, pages 160–169, April 1990.

[11] F. Olken, D. Rotem, and P. Xu. Random sampling from hash files. *SIGMOD Records*, 19(2):375–386, June 1990.

[12] Transaction Processing Performance Council. TPC-D standard specification, January 1998.